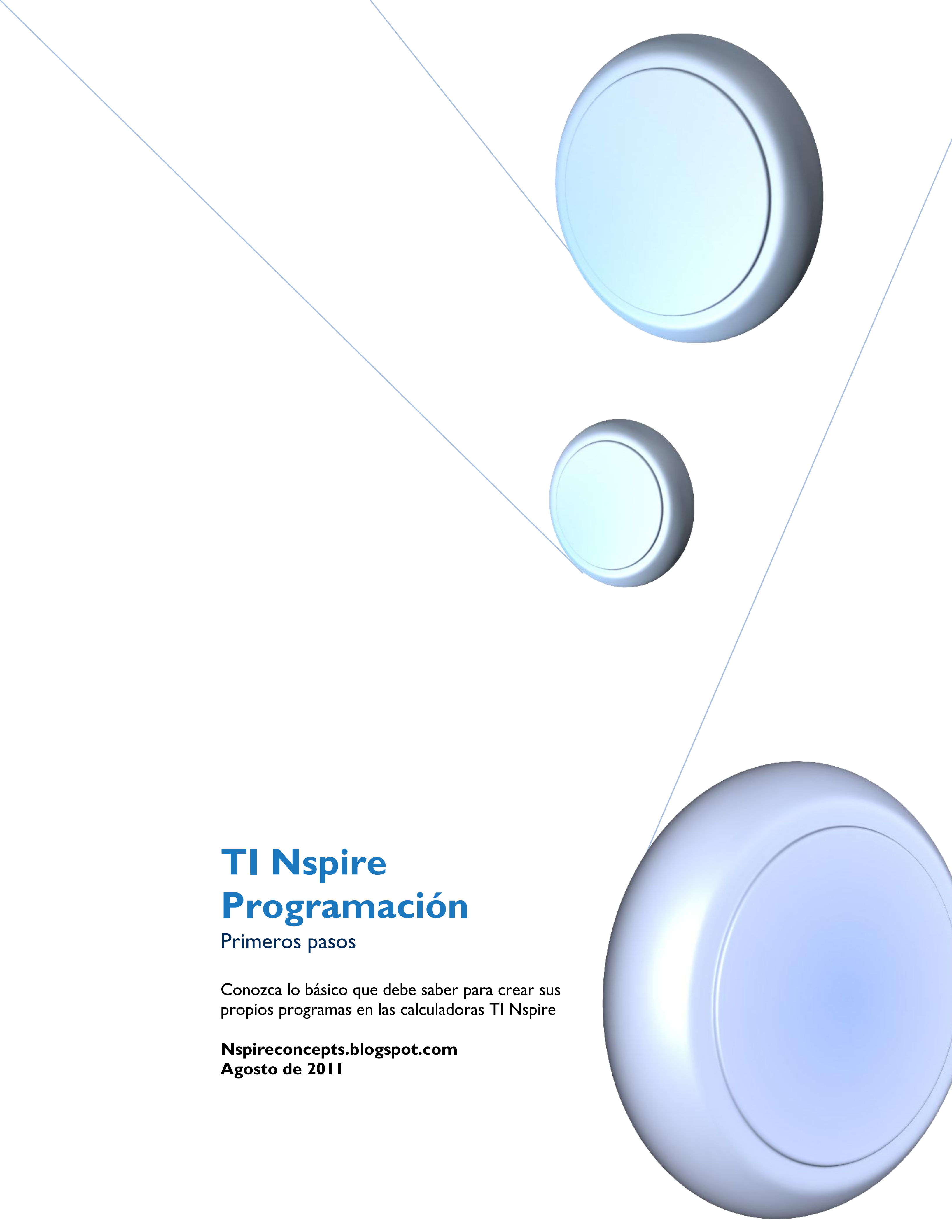


TI Nspire Programación

Primeros pasos

Conozca lo básico que debe saber para crear sus propios programas en las calculadoras TI Nspire

Nspireconcepts.blogspot.com
Agosto de 2011

TI Nspire Programación
Primeros Pasos

Contenido

1 Programas y Funciones

Este tutorial va dirigido a personas que tengan un conocimiento básico del manejo de la calculadora TI Nspire en cualquiera de sus versiones.

Hasta ahora todas las operaciones básicas como sumar, restar, multiplicar y dividir, las ha realizado en la línea de entrada de la aplicación Calculadora o en la vista Calculadora del Scratchpad. Aunque todos los comandos de la calculadora se pueden ejecutar de éste modo, la utilización de programas y funciones se justifica cuando las operaciones aumentan en cantidad y complejidad. Aunque es fácil de utilizar, la aplicación Calculadora puede llegar a no ser lo suficientemente interactiva, las operaciones y comandos no pueden volver a ser ejecutados en el mismo orden o a petición del usuario. Eso implica que cada vez que pulsa la tecla [enter] sólo se ejecuta el último comando y todo lo anterior permanece inalterado. Si desea realizar alguna corrección o cambio en alguno de los comandos previamente ejecutados, será necesario volver a escribirlos y ejecutarlos nuevamente de uno en uno.

Otra forma de ejecutar los comandos de TI Nspire, es crear un Programa o una Función con los comandos que posteriormente ejecutará. Cuando se ejecuta el programa o función, los comandos que contiene son ejecutados en el orden en que fueron escritos. Además, si fueran necesarias correcciones o cambios posteriores, sólo habría que editar el programa y ejecutarlo de nuevo.

Hay dos formas de crear programas y funciones: escribiendo las instrucciones directamente en la línea de entrada de la aplicación Calculadora o utilizando el Editor de programas.

1.1 Diferencia entre Programas y Funciones

Los programas y las funciones tienen muchas similitudes, pero sutiles diferencias.

- ☞ Una función debe retornar un resultado, que se puede graficar, o ingresar en una tabla, un programa no puede retornar resultados.
- ☞ Las funciones pueden ser utilizadas dentro de expresiones matemáticas.
- ☞ Los programas sólo se pueden ejecutar en las aplicaciones Calculadora o Notas. Las funciones pueden ser utilizadas en cualquier aplicación.
- ☞ Una función puede utilizar a otra como parte de su definición o como parte de sus argumentos. Sin embargo, una función no puede utilizar un programa como parte de su definición.
- ☞ Los programas pueden almacenar variables locales o globales. Una función sólo almacena variables locales.

1.2 Uso de variables

Una variable es un valor definido que se puede usar varias veces en un problema. Puede definir un valor o una función como una variable dentro de cada aplicación.

Dentro de un problema, las variables se comparten entre las Aplicaciones. Por ejemplo, usted puede crear una variable en la aplicación Calculadora y luego usarla o modificarla en las aplicaciones Gráficos, Geometría o en Listas y Hoja de Cálculo dentro del mismo problema.

Cada variable tiene un nombre y una definición, y la definición se puede cambiar. Cuando usted cambia la definición, todas las ocurrencias de la variable en el problema se actualizan para usar la nueva definición. Las variables son útiles y presentan ventajas como:

- ✓ Almacenar un número, si cree que podría utilizarlo en cálculos subsecuentes. Guardar un número en una variable es útil si el número es muy largo y quiere utilizar el valor completo en el futuro.
- ✓ Definir una función, las funciones y los programas pueden ser almacenados en variables.
- ✓ Almacenar una lista, cuando ésta tiene muchos elementos. Puede utilizar el nombre de la variable que guarda la lista, en lugar de volver a escribir todos los elementos dela lista.
- ✓ Almacenar una matriz, cuando ésta pueda llegar a ser muy grande, es decir, si tiene muchas filas y columnas. Guardar una matriz le ayudará a ahorrar tiempo y esfuerzo.

Vea también "Tipos de Datos".

En general utilizar variables puede ahorrarle tiempo, y también asegura que los cálculos subsiguientes sean precisos y exactos. Algunas variables son almacenadas de forma automática dependiendo de las funciones que utilice en la calculadora. Por ejemplo, cada vez que ejecuta un cálculo, el resultado se almacena en la variable Ans. Si realiza una regresión con un conjunto de datos, se crearán muchas variables.

Las variables pueden ser compartidas por aplicaciones que hagan parte del mismo problema. Es decir, si define una variable en el problema 1, sólo se puede acceder a su valor en el problema 1. Asimismo, puede definir otra variable con el mismo nombre en un segundo problema, esto no generará conflicto entre ellas. Consulte el manual de instrucciones de TI Nspire para más información acerca del "Trabajo con problemas y páginas".

Los nombres de variables, programas y funciones deben cumplir con una serie de reglas.

- ☞ El nombre de una variable puede tener desde 1 hasta 16 caracteres que pueden consistir en letras, dígitos y el carácter de guión bajo (_). Las letras pueden ser latinas o griegas, letras acentuadas y letras internacionales
- ☞ No se distinguen mayúsculas de minúsculas.

- ☞ No es posible utilizar un digito numérico como inicial del nombre de una variable
- ☞ No es posible utilizar como nombre de variable una función preasignada, como Ans, min, o tan.
- ☞ Si una variable inicia con el carácter _ se considera como un tipo de unidad como _m representa metro, _min representa minuto, etc.

Consulte el manual de instrucciones de TI Nspire para más información acerca de "Cómo nombrar variables".

2 Editor de programas

El Editor de programas es la aplicación con la cual puede trabajar con programas o funciones nuevas o existentes.

El Editor de programas tiene varias ventajas, entre ellas, tiene un menú propio que permite el acceso rápido a una variedad de plantillas y herramientas de programación. El Editor de Programas no ejecuta los comandos ni evalúa expresiones conforme usted las ingresa. Sólo se ejecutan cuando usted evalúa la función o cuando ejecuta el programa.

2.1 Abrir el Editor de programas

Hay dos formas de abrir el Editor de programas:

Desde cualquier aplicación

En el software para computador, seleccione el menú Insertar, Editor de programa, Nuevo. En la calculadora pulse [doc▾][4][A][1]: Documentos, Insertar, Editor de programa, Nuevo.

Si escribe directamente en la calculadora, o en el modo de dispositivo portátil del software TI Nspire, la página mostrará una vista dividida en la que el editor está ubicado en la sección derecha. Para desplegarlo en una página completa, pulse [ctrl][I], para insertar una página nueva, y luego pulse [doc▾][4][A] seleccione unas de las opciones Nuevo, Abrir o Importar. La opción ver muestra la definición de un programa o función, sin posibilidad de modificarla.

Desde la aplicación Calculadora

En el software para computador, seleccione el menú Insertar, Editor de programas, Nuevo. En la calculadora pulse [menu][9][1]. Menú, Funciones y Programas, Editor de programas, y seleccione unas de las opciones Nuevo, Abrir o Importar.

2.2 La ventana del Editor de programas

La vista de la aplicación Editor de programas tiene en la parte superior la Línea de estado que muestra información sobre el número de líneas y el nombre del programa o función que está editando. El asterisco al lado del nombre del programa indica que éste ha sufrido algún cambio desde la última vez que se guardó y comprobó su sintaxis. El Área de trabajo, ocupa el resto de la página, muestra la definición del programa o función.

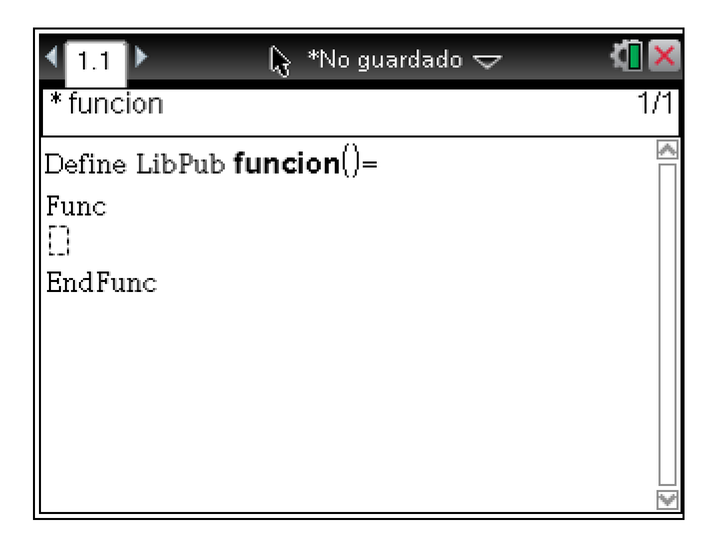


Ilustración 2.1. Vista de la aplicación Editor de programas.

Dentro del área de trabajo el comando Define (definir) siempre aparece como el primer comando de la primera línea en todos los programas. El comando LibPub (o LibPriv) establece la visibilidad del problema en una biblioteca. (Para más información acerca de éstos comandos vea Acceso a bibliotecas públicas y privadas). El nombre del programa o función aparece resaltado en negrita y otro tipo de letra con un par de paréntesis al lado, dentro de los cuales se especifica el parámetro (o los parámetros) del programa o de la función. En la segunda línea se inicia la plantilla para crear la función o programa. Cuando se crea una función ésta plantilla inicia con la palabra clave Func y termina con EndFunc en la última línea del código. En un programa la plantilla inicia con Prgm y finaliza con EndPrgm.

2.3 El menú del Editor de programas

El menú del Editor de programas puede utilizarse para introducir instrucciones de programación, plantillas, entre otras tareas.

- **Acciones**: Este submenú le permite crear nuevos programas y funciones. Es donde puede abrir, importar y ver programas, también permite crear copias, renombrar y cambiar el acceso a bibliotecas de programas o funciones existentes. Con las herramientas de edición puede insertar comentarios, buscar y remplazar texto, e ir a una línea específica.
- **Revisar sintaxis y guardar**: Aquí es donde se buscan errores de sintaxis y sitúa el cursor cerca del primer error. Si no encuentra errores, guarda el programa o función actual.
- **Definir variables**: En éste submenú es donde se definen las variables locales, insertar plantillas de función y programa.
- **Control**: Éste submenú contiene varias plantillas de bloques que le permiten controlar el flujo de un programa o función, como instrucciones condicionales y repetitivas.

- **Transferencia**: Aquí hay comandos que le permiten terminar un programa o función, saltar a otra ubicación, o alterar el flujo de las instrucciones.
- **E/S (Entrada/Salida)**: Contiene los comandos para solicitar y mostrar datos en un programa o función.
- **Modo**: Éste submenú le permite cambiar temporalmente las configuraciones de los programas o funciones, por ejemplo, puede configurar una función para que retorne valores en números binarios.

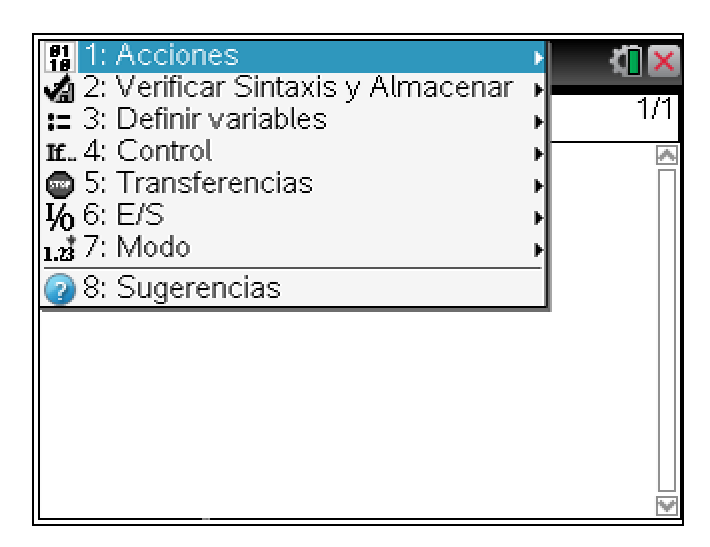


Ilustración 2.2. Vista del menú de la aplicación Editor de programas.

Consulte la "Guía de Referencia" para conocer la explicación y sintaxis de todos los comandos.

3 Programas: Primeros Pasos

Previamente se mostró cuándo y por qué utilizar programas; que además puede escribir funciones, y las diferencias entre ellos. Adicionalmente los programas tienen las siguientes ventajas:

- ✓ Un programa es una secuencia de comandos y operaciones.
- ✓ Cuando se ejecuta un programa realiza las operaciones, ejecuta los comandos y evalúa las funciones en el orden en que fueron escritos.
- ✓ Para mostrar el resultado de una operación, función o comando se debe utilizar el comando Disp, o Text.
- ✓ La utilización de programas es útil ya que estos pueden ser editados (corregidos o modificados) y se pueden ejecutar cuantas veces se desee.

3.1 Definición de un Programa

1. Abra el Editor de programas.
2. Escriba el nombre del programa.
3. Seleccione "Programa" en la lista Tipo.
4. Seleccione el tipo de acceso a biblioteca. Ninguno: para utilizar el programa sólo desde el documento. LibPriv: para utilizar el programa desde cualquier documento sin mostrarlo en el Catálogo. LibPub: para utilizar el programa desde cualquier documento y mostrándolo en el Catálogo. Vea Acceso a Bibliotecas Públicas y Privadas.
5. Haga clic en OK, o pulse [enter]. Se abrirá una página con el Editor de programas.

Ejemplo 3.1

Abra un documento nuevo, e inserte una página con la aplicación Calculadora. Defina una lista con valores diferentes (incluso palabras si utiliza la versión CAS), separados por comas.

$$lista := \{1,2,4,9,a,(x+b)^2\}$$

Defina un programa con el nombre "raíz_lista". Haga clic en OK, o pulse [enter].

Observe de debajo de la palabra clave Prgm hay un rectángulo en línea punteada, a partir de allí es donde debe introducir los comandos y operaciones para su programa; haga clic sobre él o desplácese hasta el con las teclas ▲ o ▼ , escriba la instrucción lista:=sqrt(lista).

A continuación, diríjase al menú del Editor de programas, en el submenú Verificar Sintaxis y Almacenar, escoja la opción Verificar sintaxis y almacenar. En la calculadora

pulse menu 2 1, o utilice el atajo ctrl B. Note que donde había escrito "sqrt" ahora aparece el símbolo de la raíz cuadrada

Diríjase a la aplicación Calculadora y escriba el nombre del programa, "raíz_lista" o pulse la tecla var y selecciones el programa que acaba de crear en la lista, y pulse enter. Al lado derecho de la página de la aplicación Calculadora aparece "Hecho", esto significa que el programa que acaba se ejecutó con éxito.

Pulse h y seleccione lista, pulse enter, verá como la definición de la lista que había creado previamente ha cambiado.

En el ejemplo anterior el programa realiza una operación sobre una variable llamada lista. Ésta variable es considerada como "variable global" pues su contenido está disponible para cualquier otra aplicación que se inserte en el mismo problema del documento.

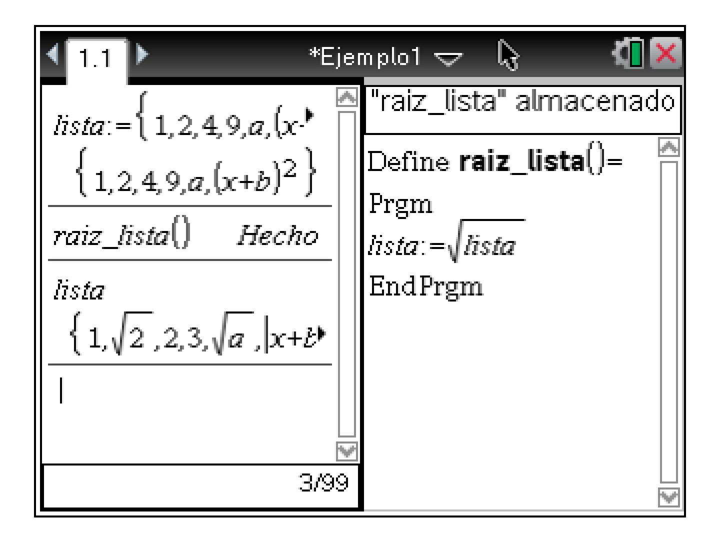


Ilustración 3.1. Ejecución del programa raiz_lista.

En la línea $lista := \sqrt{lista}$, el programa llama al comando sqrt, que calcula la raíz cuadrada de un argumento, en este caso es una variable que contiene una lista de valores. A continuación, el resultado de dicha operación (una lista con las raíces cuadradas de cada uno de los elementos de la lista inicial) es asignado a la variable "lista".

Observe que después de ejecutarse el programa, fue necesario llamar de nuevo a la variable "lista", ya que en el programa no se ejecutó ningún comando que muestre el resultado del trabajo realizado. Los programas de TI Nspire no muestran el resultado de las operaciones realizadas o comandos ejecutados, a menos que se indique explícitamente. Para ello se utiliza el comando Disp.

Ejemplo 3.2

Pulse [menu], seleccione Acciones, Crear copia… Escriba en nombre raiz_lista2. Si se dificulta la visualización puse [ctrl][6] para separar las aplicaciones en páginas individuales, desplácese hasta la página que tiene el programa raiz_lista2 y escriba la siguiente línea:

Disp "Raíz de lista:", lista

Sintaxis

Disp argumento1, argumento2, argumento3, …

El comando Disp muestra sus argumentos en el historial de la calculadora. Los argumentos se despliegan consecutivamente, separados con espacios en blanco.
Los argumentos pueden ser variables, expresiones o texto encerrado entre comillas.
Disp puede mostrar más de un argumento separándolos con comas.

Utilice el atajo [ctrl][B] para verificar la sintaxis y almacenar la nueva definición del programa. Ejecute el programa "raíz_lista" desde la aplicación Calculadora.

En los ejemplos anteriores, el programa utiliza una variable previamente definida. Esto puede representar una limitación si el programa necesita que el usuario del programa provea valores durante la ejecución del programa. Para pedir información al usuario se utiliza el comando Request

```
Define raiz_lista2()=
Prgm
lista:=sqrt(lista)
Disp "Raíz de lista:", lista
EndPrgm
```

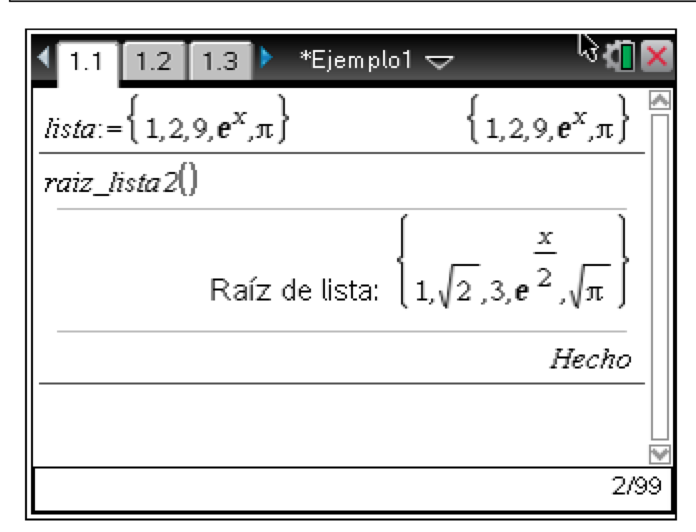


Ilustración 3.2. Ejecución del programa raiz_lista2 utilizando el comando Disp para mostrar los resultados.

Ejemplo 3.3

En el siguiente ejemplo creará un programa que pide la base y la altura de un triángulo y calcula y muestra el valor del área.

El commando Request utiliza varios parámetros, el primero es el mensaje que se muestra al usuario explicando qué información debe proveer al programa, el segundo es la variable en donde se va almacenar dicha información, y el tercero es un indicador que puede ser el número uno o cero. El tercer parámetro puede ser una expresión que se al ser evaluada produzca el mismo valor uno o cero.

```
Define tri_area()=
Prgm
Request "Base: ",b,1
Request "Altura: ",h,1
a := b*h/2
Disp "Área del triángulo: ",a
EndPrgm
```

Ilustración 3.3. Definición del programa tri_area utilizando el comando Request para pedir datos.

Sintaxis

Request textoMensaje, variable [, Bandera]

Pausa el programa y muestra un cuadro de diálogo que contiene el mensaje textoMensaje y un cuadro de entrada para la respuesta del usuario. Cuando el usuario escribe una respuesta y hace clic en OK, el contenido del cuadro de entrada se asigna a variable.
Si el usuario hace clic en Cancelar, el programa procede sin aceptar ninguna entrada.
El programa usa el valor anterior de variable si variable ya se había definido.
El argumento Bandera opcional puede ser cualquier expresión.
• Si Bandera se omite o se evalúa a 1, el mensaje de indicación y la respuesta del usuario se despliegan en el historial de la Calculadora.
• Si Bandera se evalúa a 0, la indicación y la respuesta no se muestra en el historial.

Hasta éste punto ha visto como crear un programa, a preestablecer o pedir la información necesaria para que su programa pueda trabajar con ella, y la forma como puede visualizar los resultados.

3.2 Asignación: Variables y Expresiones

En los ejemplos anteriores todos los datos introducidos y los resultados provistos al programa, se hizo mediante la asignación de variables. En el primer y segundo ejemplo se utilizó el operador de asignación (:=). En la línea a := b*h/2 se le indica a la calculadora que la variable de nombre “a” tiene un valor que es el resultado de una operación matemática, “b*h/2”.

Otra forma de asignar valores a las variables es con el símbolo de almacenar (→) cuya orden de asignación es diferente. Por ejemplo, si queremos utilizar el símbolo de almacenar tendríamos que escribir "b*h/2 → a", así, la calculadora 'entiende' que el resultado de dicha operación matemática se almacenará a la variable de nombre "a".

En ambos casos la calculadora computa la operación en caso de que las variables tengan un valor numérico previamente asignado, y luego se almacena en la variable. En la calculadora TI Nspire CAS, si alguna de las variables (o todas) no tiene un valor numérico previo, la calculadora mostrará el resultado de forma algebraica, es decir, un coeficiente acompañando a una variable.

Ejemplo 3.4

Si en el ejemplo anterior omitimos pedir el valor de la base del triángulo, el programa mostrará el resultado de la operación que calcula el área, dejando b como si fuera una incógnita.

```
Define tri_area2()=
Prgm
Request "Altura: ", h,1
 b*h/2→ a
Disp "Área del triángulo: ", a
EndPrgm

Altura: 34
El área del triángulo es 17*b
```

Ilustración 3.4. Definición y ejecución del programa tri_area2.

En las calculadoras TI Nspire (sin CAS) el programa anterior generaría un error, ya que la variable "b" aún no se ha definido con un valor numérico.

En esencia los operadores de asignación y almacenar, hacen exactamente lo mismo. La única diferencia es la notación que utilizan. Otra forma de asignar valores a las variables es con el comando Define. Aunque se utiliza con más frecuencia para la definición de programas y funciones desde la línea de entrada dela aplicación Calculadora, también se puede utilizar para definir una variable con un valor. Por ejemplo, en los programas anteriores se puede definir la variable que almacena el valor del área como:

Define a = b*h/2

Si utiliza la calculadora TI Nspire CAS, además de valores numéricos, puede almacenar expresiones dentro de variables.

Ejemplo 3.5

En una nueva página añada una aplicación Calculadora. Asigne la expresión "(x+25)*y" a la variable "z". Haga la siguiente asignación x = -20 .Escriba z y pulse [enter] para ver el contenido de la variable, verá que z es igual a 5 elevado a la potencia "y". Ahora asigne lo siguiente y = 2. Escriba z y pulse [enter]. Verá que z tiene como valor 25, ahora escriba el comando DelVar x (o pulse [menu][1][3][X]) y pulse [enter]. Borre la variable y, DelVar y. Escriba z y pulse [enter].

Durante los pasos del ejemplo anterior, se puede observar como el resultado de la expresión almacenada en la variable "z". Sin embargo, ¡la definición de z nunca se modificó! cuando borró las variables "x" y "y". Ésta característica es muy útil cuando se están creando programas y funciones, ya que permite, por ejemplo, evaluar el comportamiento de una expresión. El comando DelVar (borrar variable) elimina la definición de la variable (o las variables) que se le indique.

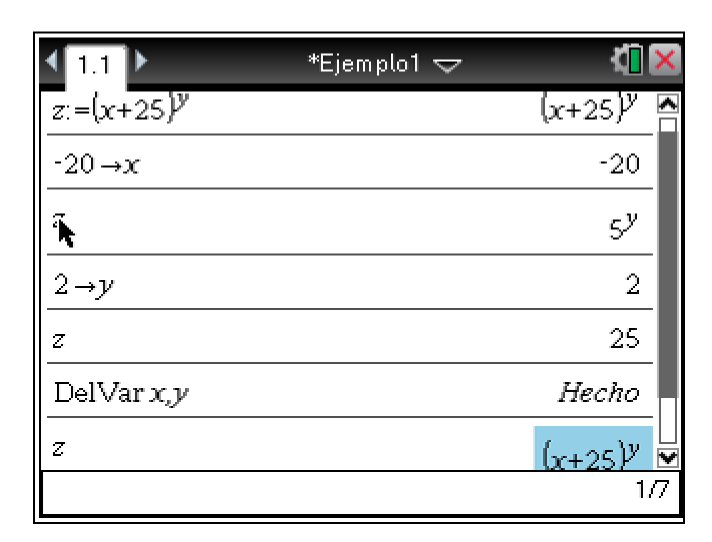


Ilustración 3.5. Asignación de valores y expresiones a variables.

Sintaxis

DelVar Var1 [, Var2] [, Var3] ...

Borra la variable o el grupo de variables especificado de la memoria. Si una o más de las variables están bloqueadas, este comando despliega un mensaje de error y borra únicamente las variables no bloqueadas

3.3 Tipos de Datos

En los ejemplos anteriores, se han utilizado listas, valores numéricos, y expresiones para almacenarlos en variables. Estos son tipos de datos, pero no son los únicos. Cada variable tiene cuatro atributos:

- Nombre definido por el usuario cuando se crea la variable.
- Ubicación en la memoria de la calculadora.
- Valor que puede ser un número, una expresión, etc.
- Tipo de dato que almacena la variable.

Tabla 3.1. Tipos de datos de TI Nspire.

Tipo de dato	Valor
Numérico	$2.34\pi \cdot (x-3)^2$
Lista	{"Amarillo", "Azul", "Rojo"}
Matriz y vector	$\begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$
Texto	"La respuesta es: "
Función	mifuncion(a0,a1,a2)
Programa	miprograma()

3.4 Variables Locales

En los primeros ejemplos se utilizaron variables globales cuyo contenido está disponible no sólo en la aplicación Calculadora, sino que además, para cualquier otra aplicación que se desee insertar en el problema del documento. Sin embargo, durante la creación de un programa puede necesitar de muchas variables, algunas de ellas no tendrán mucha relevancia una vez terminado el programa. Se puede declarar una o más variables cuya existencia permanece sólo mientras se ejecuta el programa o la función, una vez terminado el programa la variable junto con su valor desaparece. Éstas variables son llamadas Variables locales.

Sintaxis
Local Var1[, Var2] [, Var3]
Declara las variables especificadas como variables locales. Esas variables existen sólo durante la evaluación de una función y se borran cuando la función termina la

ejecución.

Al igual que las variables locales, un programa puede contener variables cuyo valor sea de vital importancia para la solución de algún problema, y podría necesitar de ese valor en futuros cálculos y operaciones. Una vez creada una variable se puede bloquear, con el objetivo de no permitir su modificación o eliminación.

3.5 Bloquear y Desbloquear variables

Para bloquear variables utilice el comando Lock (Bloquear). Para modificar o borrar una variable bloqueada, se debe desbloquear primero con el comando Unlock. Las variables desbloqueadas se muestran con un ícono de candado en la lista del menú de variables. Para saber si una variable está o no bloqueada se utiliza el comando getLockInfo.

El bloqueo de variables es útil, por ejemplo, cuando se está trabajando con listas de datos, especialmente cuando tienen un gran número de elementos. Una vez se han ingresado todos los datos en la variable, es muy recomendable bloquearla, esto no resulta un impedimento ya que la variable sí puede utilizarse en cualquier cálculo, aunque la variable con sus datos no puede ser modificada o borrada.

Sintaxis

Lock Var1[, Var2] [, Var3] ...

Bloquea las variables o el grupo de variables especificado. Las variables bloqueadas no se pueden modificar ni borrar. Usted no puede bloquear o desbloquear la variable de sistema Ans, y no puede bloquear los grupos de variables de sistema stat. o tvm.

Sintaxis

unLock Var1[, Var2] [, Var3] ...

Desbloquea las variables o el grupo de variables especificado. Las variables bloqueadas no se pueden modificar ni borrar.

Sintaxis

getLockInfo(Var) ⇒ valor

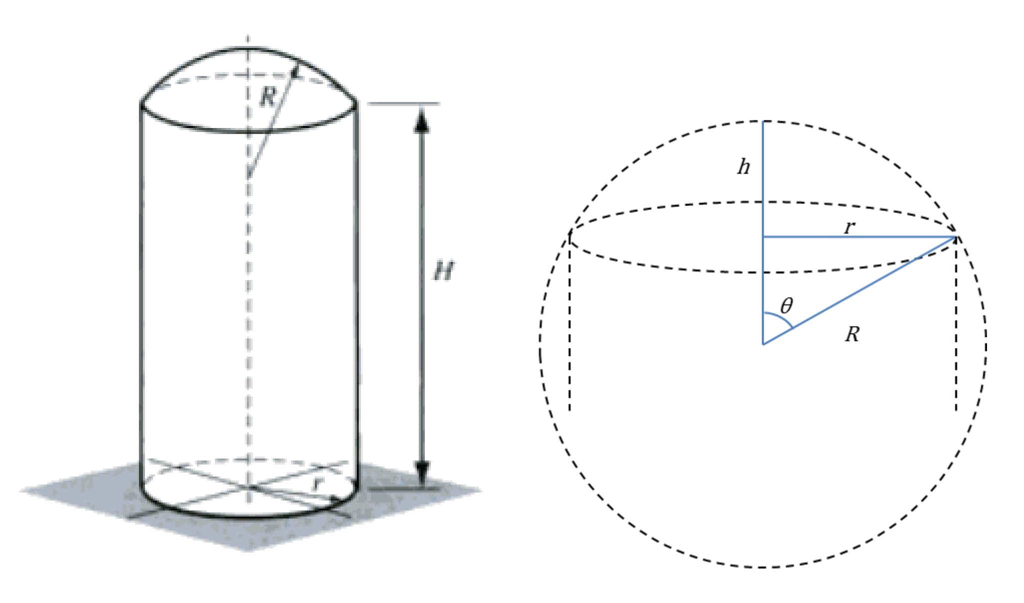
Entrega el estado de bloqueada/desbloqueada actual de la variable Var.
valor = 0: Var está desbloqueada o no existe.
valor = 1: Var está bloqueada y no se puede modificar ni borrar.

Ejemplo 3.6

Un silo con estructura cilíndrica de radio r posee una parte superior esférica de radio R. La altura de la porción cilíndrica es H. Escribir un programa que determine la altura

H a partir de los valores r y R, y el volumen V. además el programa debe calcular el área de la superficie del silo. Los datos conocidos son r = 30 pies, R = 45 pies y V = 120000 pies cúbicos.

- ☞ Asignar éstos valores directamente en la línea de entrada de la aplicación Calculadora.
- ☞ Pedir los al usuario durante la ejecución del programa.



Solución

El volumen total del silo se obtiene sumando el volumen de la parte cilíndrica y el de la parte esférica que corresponde al techo. El volumen del cilindro se calcula con la expresión

$$V_{cilindro} = \pi r^2 H$$

El volumen del techo se obtiene a partir de

$$V_{techo} = \frac{1}{3}\pi h^2(3R - h)$$

Donde $h = R - R\cos\theta = R(1 - \cos\theta\,)$ y θse obtiene de $\sin\theta = \frac{r}{R}$

La altura H de la parte cilíndrica se puede expresar de la forma

$$H = \frac{V - V_{techo}}{\pi r^2}$$

El área de la superficie del silo se calcula sumando el área de la parte esférica y el de la parte cilíndrica

$$S = S_{cilindor} + S_{techo} = 2\pi rH + 2\pi RH$$

```
Define silo_a()=
Prgm
θ:=arcsin(rc/re)
he:=re*(1-cos(θ))
v_techo:=pi*he^(2)*(3*re-he)/3
hc:=(v-v_techo)/(pi*rc^(2))
s:=2*pi*(rc*hc+re*he)
Disp "La altura del silo es: ", hc, "pies."
Disp "La superficie del silo es: ", s, "pies cuadrados."
EndPrgm
```

Ilustración 3.6. Definición del programa silo_a.

```
Define silo_b()=
Prgm
Local θ, he, v_techo, rc, re, v
Request "Radio del silo:", rc
Request "Radio de la esfera:", re
Request "Volumen del silo:", v
θ:=arcsin(rc/re)
he:=re*(1-cos(θ))
v_techo:=pi*he^(2)*(3*re-he)/3
hc:=(v-v_techo)/(pi*rc^(2))
s:=2*pi*(rc*hc+re*he)
Disp "La altura del silo es:", hc, "pies."
Disp "La superficie del silo es:", s, "pies cuadrados."
EndPrgm
```

Ilustración 3.7. Definición del programa silo_b.

4 Funciones

Una función matemática, f(x), asocia un único número (valor de la función) a cada uno de los valores de x. Las funciones se pueden expresar en la forma y=f(x), donde f(x) es habitualmente un expresión matemática en términos de la variable x. Cuando se conoce el valor de x (entrada) en la expresión de la función, se obtiene un valor y (salida). TI Nspire posee muchas funciones preprogramadas que pueden ser utilizadas en expresiones simplemente escribiendo su nombre junto con el argumento (o argumentos) de entrada; por ejemplo, sin(x), sqrt(x), etc.

Frecuentemente, a la hora de programar existe la necesidad de operar con funciones distintas a las predefinidas. Cuando la expresión de la función es sencilla se puede incluir en la línea de entrada de la aplicación Calculadora. Sin embargo, cuando la expresión de la función tiene mayor complejidad o condiciones especiales, es conveniente crearla en el Editor de programas. Una vez creada la función, ésta puede ser utilizada como cualquier otra función predefinida.

La función puede contener expresiones matemáticas simples o códigos que impliquen cálculos más complejos. La principal característica de una función es que debe tener al menos un argumento de entrada y un argumento de salida. La entrada de una función puede estar compuesta por una o más variables, y cada una de ellas pueden ser valores numéricos, matrices, listas, expresiones, otras funciones pero no programas. En muchos casos las funciones se pueden incorporar como "sub-funciones" dentro funciones o programas. De ésta forma se pueden construir programas grandes uniendo bloques más pequeños que se pueden evaluar y depurar independientemente.

4.1 Definición de una Función

1. Abra el Editor de programas
2. Escriba el nombre del programa.
3. Seleccione "Función" en la lista Tipo.
4. Seleccione el tipo de acceso a biblioteca. Ninguno: para utilizar el programa sólo desde el documento. LibPriv: para utilizar el programa desde cualquier documento sin mostrarlo en el Catálogo. LibPub: para utilizar el programa desde cualquier documento y mostrándolo en el Catálogo. Vea Acceso a Bibliotecas Públicas y Privadas.
5. Haga clic en OK, o pulse [enter]. Se abrirá una página con el Editor de programas.

4.2 Argumentos

Los argumentos de entrada y salida se utilizan para transferir datos hacia dentro y hacia fuera de la propia función. Los argumentos de entrada se introducen entre paréntesis a continuación del nombre de la función. Si la función se piensa para más de un parámetro, éstos deben ir separados por comas. Normalmente el código

interno que contiene la función está pensado para operar sobre los argumentos de entrada, y por tanto se presupone que estos deben ser valores apropiados y en el orden correcto.

4.3 Variables locales

Todas las variables de una función son locales; cualquier variable, incluidos los argumentos de entrada y salida se pueden utilizar dentro del código de la función. Esto significa que las variables definidas dentro de la función sólo se existirán dentro de ella. En una función se asignan nuevos valores a las variables de entrada cada vez que se invoca la función. Cuando la función termina su ejecución, el valor del argumento de salida es transferido a la variable que se le asignó en el momento en el que la función fue invocada, si no se asignó a ninguna variable el argumento de salida se transfiere a la variable Ans.

Una función no modifica el valor de una variable que se haya definido fuera de ella, es decir, se puede definir una variable fuera de la función y dentro de ella definir otra con el mismo nombre. La función solo trabajará sólo con la variable que haya sido definida en el interior de su código. La variable externa no sufrirá ningún cambio. Esto significa que todos los valores con los que trabaja la función deben ser ingresados como argumentos o generados por alguna operación dentro del cuerpo de la función.

4.4 Funciones en línea

Las funciones en línea se pueden utilizar para construir funciones matemáticas con expresiones sencillas. En general se usan en casos en los que un valor relativamente simple debe obtenerse varias veces durante una serie de cálculos o dentro de un programa (inclusive en una función más compleja). Una función de línea se pueden definir en la línea de entrada de la aplicación Calculadora (de ahí su nombre) o dentro de un programa.

Características

- La expresión puede tener una o más variables independientes.
- La expresión puede incluir cualquier función predefinida TI Nspire, u otra creada por el usuario.
- Una función en línea también se puede utilizar como argumento de otra función.

5 Bibliotecas

Las bibliotecas son documentos que están ubicados en la carpeta MyLib de Mis documentos. Las bibliotecas contienen variables, funciones o programas que están disponibles para ser utilizadas por cualquier documento. Estos son diferentes de los programas y variables ordinarias ya que están solo disponibles para un solo problema en un solo documento.

Si debe definir las mismas funciones, almacenar las mismas variables, o realizar los mismos pasos en cálculos repetidamente, podría considerar definir una Biblioteca.

Definiendo un programa como parte de una biblioteca, estará siempre disponible para ser utilizado por cualquier documento, no sólo en el que fue creado.

5.1 Crear una biblioteca

1. Abra un documento nuevo, y defina una o más objetos de biblioteca (variable, programa o función). Ver Definición de un programa.
2. Guarde el documento en la carpeta MyLib.
3. Abra o cree un documento nuevo.
4. Pulse [doc▾] [6] Actualizar bibliotecas (también puede pulsar [menu] [1] [7] [1])
5. Pulse la tecla [📖] para abrir el Catálogo, pulse 5 (6 en CAS.)
6. Marque la Biblioteca en la lista y seleccione el programa.
7. Haga clic en OK, o pulse [enter] para insertar el programa en la aplicación.

5.2 Acceso a bibliotecas públicas y privadas

Cuando se define un objeto de biblioteca debe ser designado como privada (LibPriv) o pública (LibPub). Un objeto de biblioteca privada no aparece en el Catálogo, pero puede acceder a él al escribir su nombre. Los objetos privados sirven bien como bloques de construcción que realizan tareas básicas de bajo nivel. Por lo general, los objetos privados se requieren por las funciones o los programas públicos.

Un objeto de biblioteca pública aparece en la pestaña de biblioteca del Catálogo después de que se actualizan las bibliotecas. Se puede acceder a un objeto de biblioteca a través del Catálogo o al escribir su nombre.

6 Programación en TI Nspire: TI NSPIRE BASIC

Al igual que un programa para computador, un programa o una función de TI Nspire es una secuencia de comandos, también denominados instrucciones. En un programa sencillo los comandos se ejecutan uno tras otro según el orden en que fueron escritos. Hasta ahora todos los programas y funciones que se han presentado son códigos sencillos. Sin embargo, en muchas otras situaciones es necesario escribir programas más complejos cuyas instrucciones no tienen que ejecutarse en el mismo orden en el que se escribieron, o que se ejecuten distintos tipos de instrucciones o grupos de instrucciones en función del valor de una serie de variables del programa.

6.1 Control de programas

TI Nspire proporciona diferentes instrucciones que se pueden utilizar para controlar el flujo de un programa. Instrucciones condicionales permiten saltarse comandos o ejecutar grupos específicos de comandos en diferentes situaciones. Las instrucciones de iteración o repetición permiten repetir secuencias de comandos varias veces seguidas.

Para cambiar el flujo de un programa requiere algún tipo de proceso de toma de decisiones dentro del programa, éste debe decidir si ejecuta el siguiente comando o pasa por alto uno o varios comandos y continuar en otra línea de código distinta. El programa toma éstas decisiones comparando los valores de las variables. Esto se lleva a cabo mediante los operadores lógicos y de relación.

6.1.1 Operadores de relación

Un operador de relación compara dos expresiones determinando si el enunciado de la comparación es verdadero o falso.

Los operadores de relación de TI Nspire son los siguientes:

Tabla 6.1. Descripción de los operadores de relación de TI Nspire.

Operador	Descripción
>	Mayor que
<	Menor que
≥	Mayor o igual que
≤	Menor igual que
=	Igual que
≠	Diferente a

6.1.2 Operadores lógicos

Los operadores lógicos proporcionan un resultado a partir de que se cumpla o no una cierta condición.

Tabla 6.2. Descripción de los operadores lógicos de TI Nspire.

Operador	Ejemplo	Descripción
And	X and Y	Funciona con dos operandos. El resultado es verdadero (true) si ambos son verdaderos. En caso contrario el resultado es falso (false).
Or	X or Y	Funciona con dos operandos. El resultado es verdadero (true) si alguno de los dos es verdadero. Si los dos son falsos el resultado será falso (false)
Not	not Z	Funciona con un operando. Da la negación del operando, es decir, verdadero si Z es falso, y falso si Z es verdadero.

6.2 Instrucciones condicionales

Una instrucción condicional permite tomar decisiones sobre si se ejecuta o no un grupo de comandos que cumplen con una o varias condiciones.

6.2.1 Instrucción condicional simple: If

Los programas utilizan instrucciones condicionales para tomar decisiones sobre si se ejecuta un grupo de comandos que cumplen una condición, o por el contrario, omitirlos. En ésta estructura se evalúa la expresión. Si es verdadera el grupo de comandos se ejecutará. Si la expresión es falsa, el programa saltará el grupo de comandos en cuestión. La forma básica de la instrucción if es

```
If condición Then
  Bloque
EndIf
```

Si la condición se evalúa como verdadera, ejecuta una instrucción sencilla o un bloque de instrucciones antes de continuar con la ejecución. Si la condición se evalúa como falsa, continúa con la ejecución sin ejecutar la instrucción o el bloque de instrucciones.

```
If condición
  Instrucción
```

Ésta estructura ejecuta sólo una instrucción si la condición se evalúa como verdadera. Si la condición se evalúa como falsa, se omite esa instrucción de la ejecución.

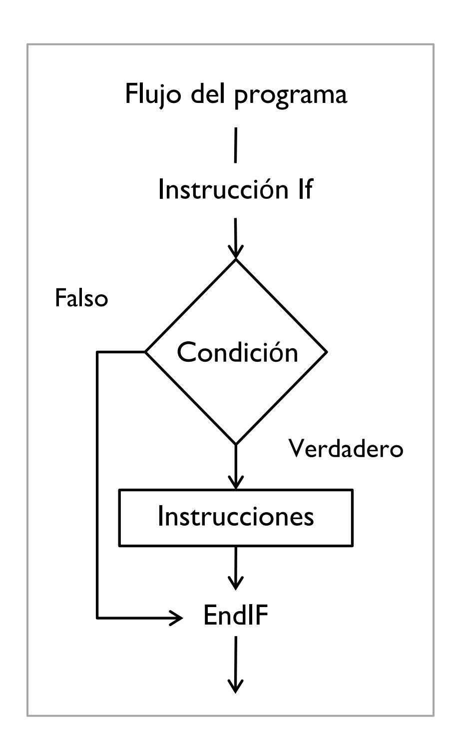


Diagrama 6.1. Estructura del flujo de programa con una instrucción condicional If.

Ejemplo 6.1

Un trabajador cobra un determinado salario por hora hasta 40 horas semanales. Además, si hace horas extra (más de 40 horas semanales), esas horas se las pagan 50% más. Escribir un programa que calcule el sueldo semanal del trabajador. El programa pedirá al usuario que introduzca las horas trabajadas y el valor de la hora.

El cálculo del sueldo semanal es simplemente multiplicar la cantidad de horas por el valor de la hora trabajada. Sin embargo cuando se han trabajado más de 40 horas a la semana, cada hora extra vale 50% más. Para que el programa determine si las horas trabajadas fueron mayores a 40 se compara una variable que almacena la cantidad de horas con 40, si es mayor, al producto horas × valor se sumará las horas extra por la mitad del valor de una hora normal.

```
Define pagosueldo()=
Prgm
Local t, h, pago
Request “Horas trabajadas:”, t
Request “Valor hora trabajada: $”, h
Pago := t*h
If t>40
  pago := pago + (t-40)*0.5*h
Disp “El pago seminal es: $”, pago
```

```
EndPrgm
```

Ilustración 6.1. Definición del programa pagoSueldo.

6.2.2 Instrucción condicional doble: If - Else

Esta estructura le permite ejecutar uno entre dos grupos o bloques de instrucciones en función de la evaluación de una condición. Se distingue una entre dos opciones posibles, al contrario que la instrucción anterior donde sólo era posible ejecutar o no un bloque de instrucciones.

```
If condición Then
  Bloque 1
Else
  Bloque 2
EndIf
```

Si la condición se evalúa como verdadera, ejecuta el bloque 1 de instrucciones antes de continuar con la ejecución. Si la condición se evalúa como falsa, ejecuta el bloque 2 de instrucciones.

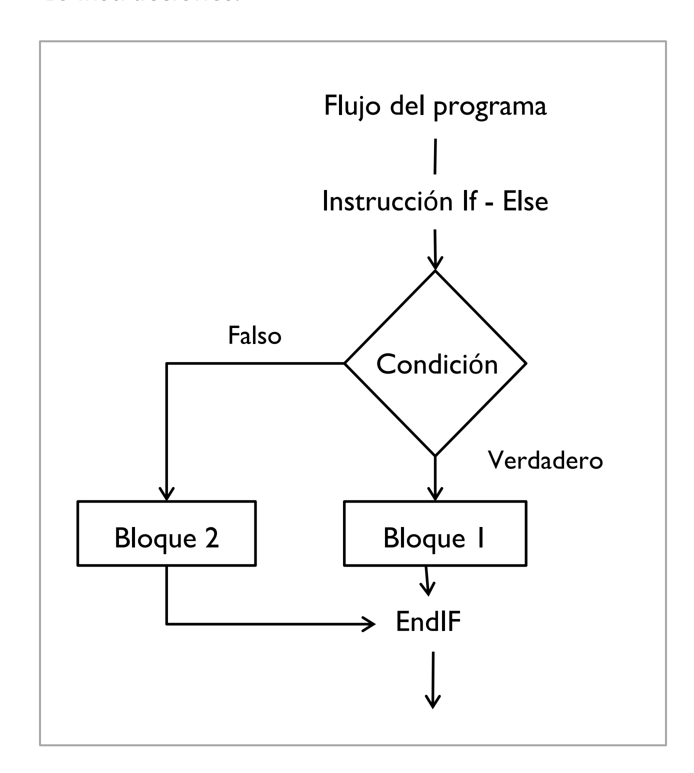


Diagrama 6.2. Estructura del flujo de programa con una instrucción condicional If – Else.

Ejemplo 6.2

Una torre para el almacenamiento de agua tiene la geometría que se muestra en la figura, en la parte inferior es un cilindro y la superior un cono truncado invertido. Dentro del depósito hay una boya que indica el nivel del agua. Escribir una función que calcule el volumen de agua dentro del depósito a partir de la posición (altura h) de la boya. La entrada de la función será el valor de la altura h en metros, y la salida será el volumen que ocupa el agua en metros cúbicos.

Solución

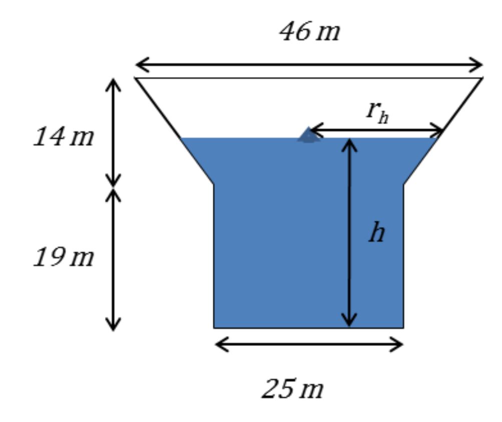
Para valores de h entre 0 y 19 metros el volumen del agua se puede calcular a partir del volumen de un cilindro de altura h:

$$V = \pi \cdot 12.5^2 \cdot h$$

Para valores de h entre 19 y 33 metros el volumen de agua se calcula sumando el volumen del cilindro y el volumen del agua en el cono:

$$V = \pi \cdot 12.5^2 \cdot h + \frac{1}{3}\pi\left(12.5^2 + 12.5 \cdot r_h + r_h^2\right)$$

Donde $r_h = 12{,}5 + \frac{10.5}{14}(h - 19)$




```
Define volagua(h)=
Func
© Calcula el volumen de agua de un depósito.
Local v
If h<=19 Then
  v := pi*(12.5)^2 *h
Else
  rh := 12.5 + (10.5*(h-19))/14
  v:= pi*12.5^2*19+pi*(h-19)*(12.5^2 + 12.5*rh+rh^2)
```

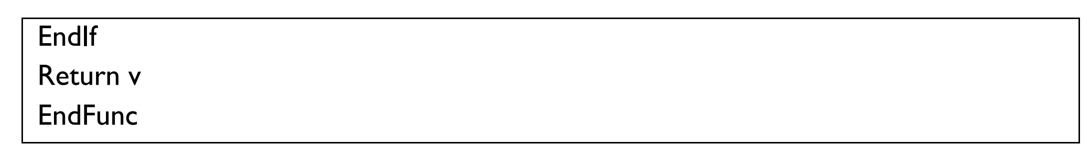

```
EndIf
Return v
EndFunc
```

Ilustración 6.2. Definición de la función volagua.

6.2.3 Instrucción condicional multiple: If – ElseIf – Else

La estructura condicional múltiple incluye dos o más condiciones, lo que hace posible ejecutar uno de estos grupos o bloques de instrucciones diferentes. Se distingue una entre dos o más opciones posibles todas sujetas a condiciones.

La primera línea es una sentencia If con condición 1. Si la condición se evalúa como verdadera, se ejecutará el bloque 1 de instrucciones y continuará con la ejecución del programa saltando las demás condiciones. Si condición 1 se evalúa como falsa, el programa saltará a la siguiente sentencia ElseIf. Si la condición 2 se evalúa como verdadera se ejecuta el bloque 2. Si la condición 2 se evalúa como falsa el programa saltará la siguiente sentencia ElseIf. Si en ninguna de las sentencias ElseIf se evalúa la condición como verdadera, el programa saltará a la sentencia Else y ejecutará el bloque final de instrucciones.

```
If condición 1 Then
  Bloque 1
ElseIf condición 2 Then
  Bloque 2
Else
  Bloque final
EndIf
```

Esta estructura también representa un mecanismo para escoger y ejecutar un grupo de instrucciones entre varios grupos posibles. Lo anterior significa que también puede utilizarse como una estructura de selección ya que se pueden utilizar tantas instrucciones ElseIf se necesite y la última instrucción Else con el bloque final pueden ser opcionales.

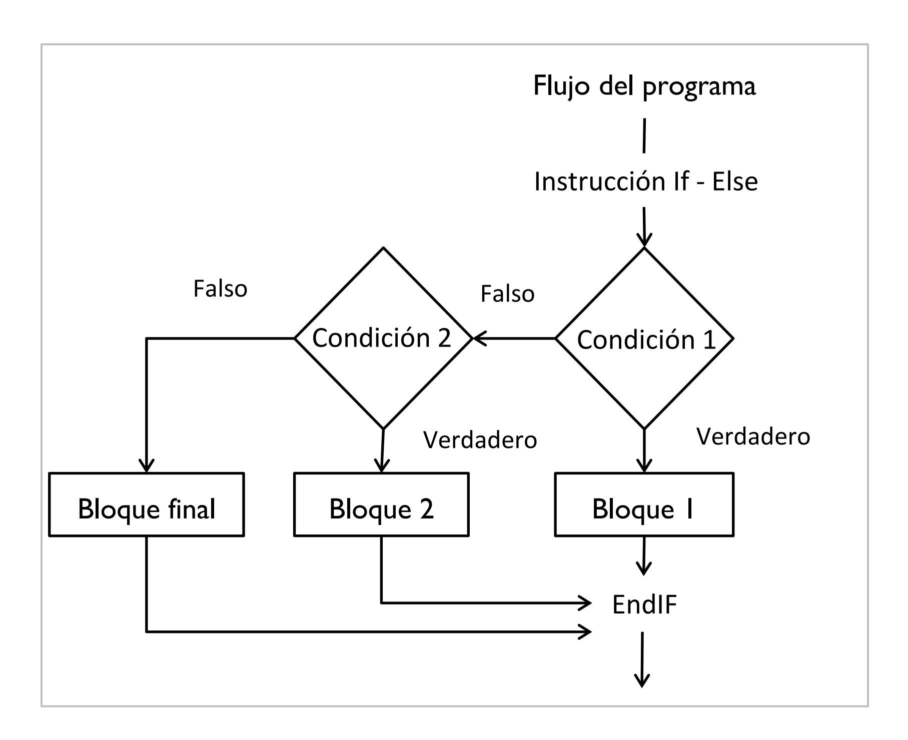


Ilustración 6.3. Estructura del flujo de programa con una instrucción condicional If – ElseIf - Else.

Ejemplo 6.3

Escribir un programa que convierta una cantidad de energía dada en Joules, pies-libra, calorías o electronvoltio, a la cantidad equivalente en otra unidad diferente especificada por el usuario. El programa pedirá al usuario que introduzca la cantidad de energía y su unidad, así como la unidad a la que se quiere realizar la conversión. La salida será la cantidad introducida convertida a la nueva unidad de energía elegida.

Los factores de conversión son I J = 0.738 ft-lb = 0.239 cal = 6.24×1018 eV

```
Define energia()=
Prgm
©Convertir unidades de energía
Local ein, einu, eout, eoutu, error, ej
error:=false
Request "Valor de energía (trabajo):", ein
Text "J: joule, ftlb: libra pie, cal: caloría, eV: electronvoltio ",0
RequestStr "Unidad actual de energía:", einu
RequestStr "Nueva unidad de energía:", eoutu
© A continuación se convierte la cantidad de entrada a Joule
If einu="J" or einu="j" Then
  ej:=ein
ElseIf einu="ftlb" Then
```

```
  ej:=ein/0.738
ElseIf einu="cal" Then
  ej:=ein/0.239

ElseIf einu="eV" or einu="ev" Then

  ej:=ein/6.24E18
Else
  error:=true
EndIf
© A continuación se realiza la última conversión
If eoutu="J" or eoutu="j" Then
  eout:=ej
ElseIf eoutu="ftlb" Then
  eout:=ej*0.738
ElseIf eoutu="cal" Then
  eout:=ej*0.239
ElseIf eoutu="eV" or eoutu="ev" Then
  eout:=ej*6.24□18
Else
  error:=true
EndIf

If error=true Then
  Text "Error, la unidad de entrada o de salida incorrecta"
Else
  Disp "E = ",eout,eoutu
EndIf
EndPrgm
```

Ilustración 6.4. Definición del programa energía, utilizando las instrucciones If – ElseIf – Else como estructura de selección.

Sintaxis

RequestStr textoMensaje, variable,[bandera]

Opera de la misa forma que el comando Request, excepto que la respuesta del usuario siempre se interpreta como una cadena de texto. Pausa el programa y muestra un cuadro de diálogo que contiene el mensaje textoMensaje y un cuadro de entrada para la respuesta del usuario. Cuando el usuario escribe una respuesta y hace clic en OK, el contenido del cuadro de entrada se asigna a variable.
Si el usuario hace clic en Cancelar, el programa procede sin aceptar ninguna entrada. El programa usa el valor anterior de variable si variable ya se había definido.
El argumento Bandera opcional puede ser cualquier expresión.
• Si Bandera se omite o se evalúa a 1, el mensaje de indicación y la respuesta del usuario se despliegan en el historial de la Calculadora.

• Si Bandera se evalúa a 0, la indicación y la respuesta no se muestra en el historial.

Sintaxis
Text textoMensaje, [bandera]
Pausa el programa y despliega la cadena de caracteres textoMensaje en un cuadro de diálogo. La ejecución del programa continúa cuando el usuario selecciona OK. El argumento bandera opcional puede ser cualquier expresión. • Si bandera se omite o se evalúa como 1, el mensaje de texto se agrega al historial de la Calculadora. • Si bandera se evalúa a como 0, el mensaje de texto no se agrega al historial.

Sintaxis
© Texto de Comentario
Procesa el contenido del texto como una línea de comentario. © puede estar al principio o en cualquier lugar de la línea. Cualquier contenido situado a la derecha del símbolo ©, y hasta el final de la línea se considera como comentario. Los comentarios son útiles para introducir anotaciones en el programa, además, no afectan el funcionamiento del programa ni aparecen cuando se ejecuta. En los programas y funciones de biblioteca pública puede definir un comentario de ayuda que será mostrado en el catálogo, escribiéndolo inmediatamente después de la línea Prgm o Func. Éste comentario puede servir para explicar la sintaxis del programa o la función

6.3 Instrucciones iterativas o de repetición

Las instrucciones iterativas permiten a los programas o funciones ejecutar instrucciones en forma repetida, siempre y cuando una condición siga siendo verdadera después de cada repetición.

6.3.1 Instrucción de repetición For

En éste tipo de estructura la ejecución de una o varias instrucciones se repite un número fijo de veces. La forma básica de la instrucción For es:

```
For contador, inicio, final, paso
  Bloque
EndFor
```

Se ejecuta el bloque de instrucciones repetitivamente cada vez que el valor de la variable contador, desde inicio hasta final en incrementos de paso. Contador puede ser cualquier nombre que no sea una variable del sistema. Paso puede ser un valor positivo o negativo. Si se omite el paso, de forma predeterminada se toma como 1.

En el primer paso el contador toma el valor de inicio, y se ejecutan las instrucciones dentro del bloque. A continuación el programa vuelve al comienzo (la instrucción

For) para realizar el segundo paso, ahora contador incrementa en un valor paso (contador = contador + paso) y el bloque de instrucciones se ejecuta de nuevo. El proceso se repite hasta que contador tiene el valor de final.

Ejemplo 6.4

Escriba una función que calcule el seno de x que puede aproximarse utilizando la serie de Taylor.

$$\sin(x) = \sum_{n=0}^{\infty} (-1)^n \frac{x^{2n+1}}{(2n+1)!}$$

Los argumentos son el ángulo θ y el número de términos de la serie.

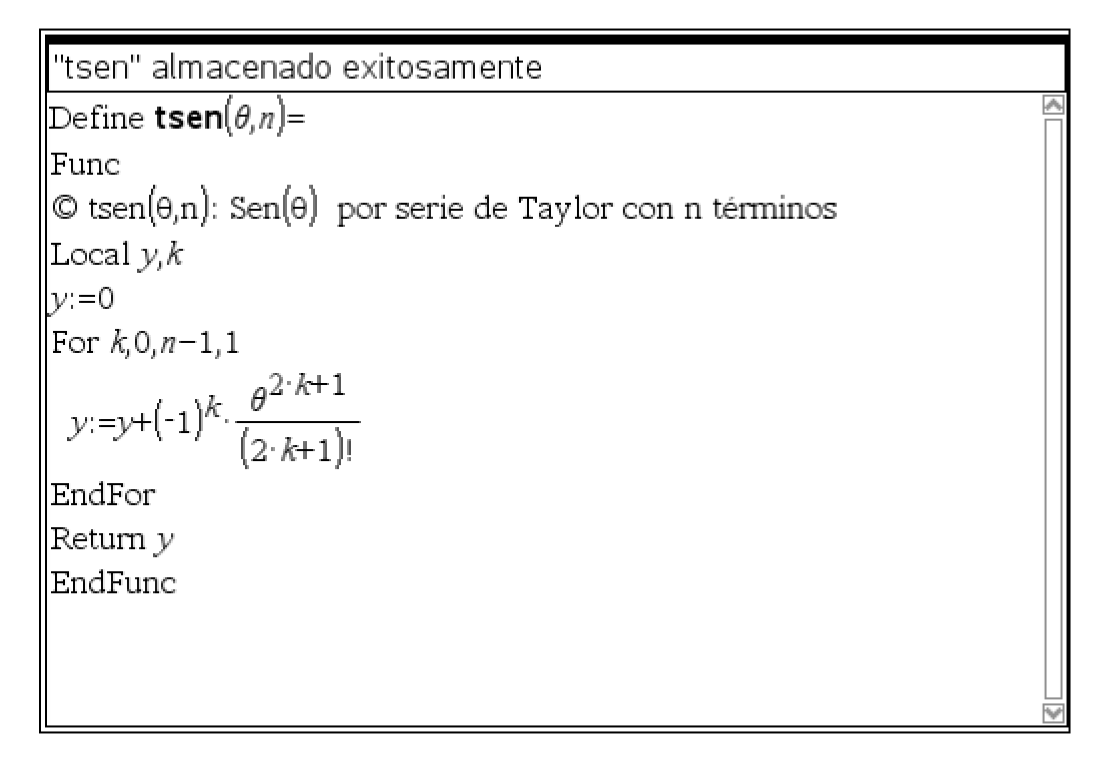


Ilustración 6.5. Definición de la función Seno aproximada por serie de Taylor.

6.3.2 Instrucción de repetición While

Esta estructura de repetición se utiliza cuando se necesita un proceso iterativo, pero se desconoce previamente el número de iteraciones que se deben realizar. En éste tipo de repetición permite especificar que un programa debe repetir un conjunto de instrucciones mientras una condición sea verdadera. La estructura básica de la instrucción while es:

```
While Condición
  Bloque
```

EndWhile

Cuando el programa llega a la primera línea de While evalúa la condición, si resulta falsa se omite la ejecución de toda la estructura. Si la condición se evalúa como verdadera se ejecuta el bloque de instrucciones y el programa regresa al inicio de la estructura para volver a evaluar la condición tantas veces ésta sea verdadera. En algún momento, la condición será falsa, en éste punto la repetición terminará y se ejecutará la primera instrucción que esté después de la estructura While.

Las variables que hagan parte de la expresión de condición deben tener valores asignados cuando se ejecute While por primera vez. Adicionalmente, al menos una de las variables de la condición debe ser modificada durante las repeticiones, de forma que la condición se haga falsa en algún momento y el proceso iterativo pueda terminar. En caso contrario, el ciclo continuará indefinidamente porque la condición de la iteración siempre es verdadera.

Ejemplo 6.5

La función exponencial puede aproximarse mediante la serie de Taylor

$$e^x = \sum_{n=0}^{\infty} \frac{x^n}{n!}$$

Escribir un programa que calcule e^x sumando los términos de la serie en un proceso iterativo que se detenga cuando el valor absoluto del último término sumado sea inferior a 0.0001. Si en la iteración número 30 el término no es inferior a 0.0001, el programa deberá parar y visualizar un mensaje que diga que se necesitan más de 30 términos.

Solución

```
Define texp()=
Prgm
© texp() calcula exp(x) con serie de Taylor hasta con 30 términos
Request "Digite el valor de x:", x
n:=1
an:=1
s:=an
While n≤30 and abs(an)≥0.0001 © Inicio del ciclo
  an:=x^(n)/(n!)    © Cálculo del enésimo término
  s:=s+an           © Se suma el enésimo término a la suma acumulada
 n:=n+1             © Se cuenta el número de iteraciones
EndWhile
```

```
If n≥30 Then
  Text "Se necesitan más de 30 términos"
Else
  Local info1, info2
  info1:="exp(" & string(x) & ") = " & string(s)
  Text info1
  info2:="Se utilizaron " & string(n) & " términos"
  Text info2
EndIf
EndPrgm
```

Ilustración 6.6. Definición del programa tExp, que calcula la función exponencial con la serie de Taylor.